CPE403 – Advanced Embedded Systems

Design Assignment 1

Name: Jenifer Christina

Email: chrisj14@unlv.nevada.edu

Github Repository link: https://github.com/chrisj14/CCS-Assignment

Youtube Playlist link: https://youtu.be/cwDEEj2b8To

Follow the submission guideline to be awarded points for this Assignment.

1. Code for Tasks. for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only. Use separate page for each task.

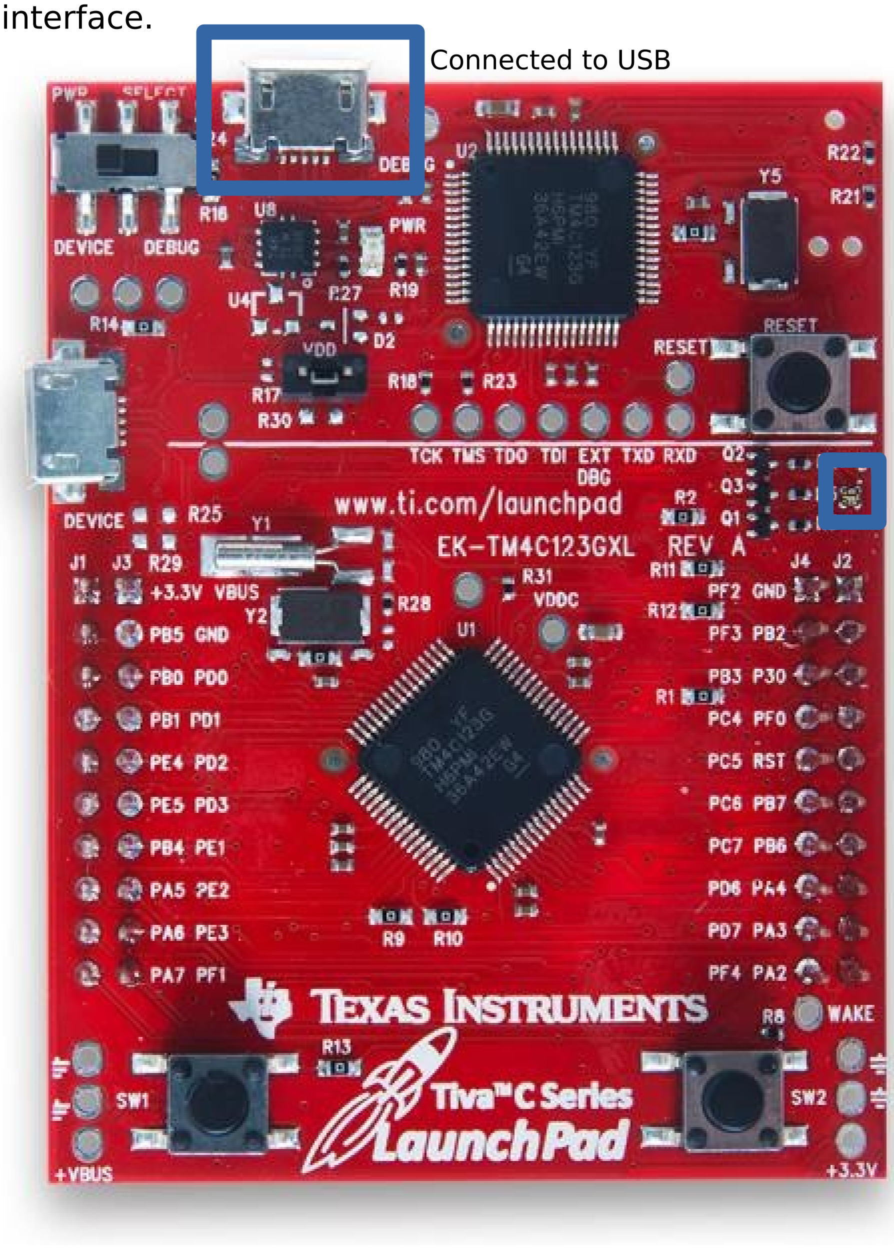
Task 03: Continue with Task 02, implement the temperature-memory transfer and memory-UART transfer using uDMA.

```
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "inc/hw_ints.h"
#include "inc/tm4c123gh6pm.h"    //def. for the interrupt and register
assignments on the Tiva C Series device on the launchPad board
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/udma.h"
#include "driverlib/timer.h"    //Defines and macros for Timer API of
driverLib.
#include "driverlib/adc.h"
#include "driverlib/debug.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
```

```c
// For Temperature value
uint32_t ui32Period;
char buffer [4];
uint32_t ui32ADC0Value[4];
volatile uint32_t ui32TempAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

// Define source and destination buffers
#define MEM_BUFFER_SIZE         1024
static uint32_t g_ui32SrcBuf[MEM_BUFFER_SIZE];
static uint32_t g_ui32DstBuf[MEM_BUFFER_SIZE];

// Define errors counters
static uint32_t g_ui32DMAErrCount = 0;
static uint32_t g_ui32BadISR = 0;

// Define transfer counter
static uint32_t g_ui32MemXferCount = 0;

// The control table used by the uDMA controller.  This table must be aligned
to a 1024 byte boundary.
#pragma DATA_ALIGN(pui8ControlTable, 1024)
uint8_t pui8ControlTable[1024];

// Library error routine
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

// uDMA transfer error handler
void
uDMAErrorHandler(void)
{
    uint32_t ui32Status;

    // Check for uDMA error bit
    ui32Status = uDMAErrorStatusGet();

    // If there is a uDMA error, then clear the error and increment the error
counter.
    if(ui32Status)
    {
        uDMAErrorStatusClear();
        g_ui32DMAErrCount++;
    }
}

// uDMA interrupt handler. Run when transfer is complete.
void
uDMAIntHandler(void)
```

```
{
    uint32_t ui32Mode;

    // Check for the primary control structure to indicate complete.
    ui32Mode = uDMAChannelModeGet(UDMA_CHANNEL_SW);
    if(ui32Mode == UDMA_MODE_STOP)
    {
        // Increment the count of completed transfers.
        g_ui32MemXferCount++;

        // Configure it for another transfer.
        uDMAChannelTransferSet(UDMA_CHANNEL_SW, UDMA_MODE_AUTO,
                                   g_ui32SrcBuf, g_ui32DstBuf,
MEM_BUFFER_SIZE);

        // Initiate another transfer.
        uDMAChannelEnable(UDMA_CHANNEL_SW);
        uDMAChannelRequest(UDMA_CHANNEL_SW);
        uint32_t ui32Status;

            ui32Status = UARTIntStatus(UART0_BASE, true); //get interrupt
status

            UARTIntClear(UART0_BASE, ui32Status); //clear the asserted
interrupts

            while(UARTCharsAvail(UART0_BASE)) //loop while there are chars
            {
                char cChar=UARTCharGet(UART0_BASE);
                UARTCharPutNonBlocking(UART0_BASE, cChar); //echo character
                if (cChar=='R') {       //Turn on RED LED
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);
                }
                else if (cChar=='r') {  //Turn off RED LED
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
                }
                else if (cChar=='G') {  //Turn on Green LED
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
                }
                else if (cChar=='g') {  //Turn off Green LED
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
                }
                else if (cChar=='B') {  //Turn on Blue LED
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);
                }
                else if (cChar=='b') {  //Turn off Blue LED
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
                }
                else if (cChar=='T') {  //Show Temperature in Centigrade
                    ADCIntClear(ADC0_BASE,2);
                    ADCProcessorTrigger(ADC0_BASE, 2);

                    ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);
                    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
                    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) /
4096)/10;
                    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
                    UARTprintf("\n C %3d\t \n",ui32TempValueC );
                }
                else if (cChar=='t') {  //Show Temperature in Farenheit
                    ADCIntClear(ADC0_BASE,2);
                    ADCProcessorTrigger(ADC0_BASE, 2);

                    ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0Value);

                    ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] +
ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
                    ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) /
4096)/10;
                    ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
                    UARTprintf("\n F %3d\t \n",ui32TempValueF );
                }
              else if (cChar=='S') {  //Show LED Status
                    UARTprintf("\n");
                    if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_1))
                        UARTprintf("Red LED is on \n");
                    if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
                        UARTprintf("Blue LED is on \n");
                    if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_3))
                        UARTprintf("Green LED is on \n");
                }
            }
    }

    // If the channel is not stopped, then something is wrong.
    else
    {
        g_ui32BadISR++;
    }
}

// Initialize the uDMA software channel to perform a memory to memory uDMA
transfer.
void
InitSWTransfer(void)
{
    uint32_t ui32Idx;

    // Fill the source memory buffer with a simple incrementing pattern.
    for(ui32Idx = 0; ui32Idx < MEM_BUFFER_SIZE; ui32Idx++)
    {
        g_ui32SrcBuf[ui32Idx] = ui32Idx;
    }

    // Enable interrupts from the uDMA software channel.
    IntEnable(INT_UDMA);
    // Place the uDMA channel attributes in a known state. These should
already be disabled by default.
    uDMAChannelAttributeDisable(UDMA_CHANNEL_SW,
                                    UDMA_ATTR_USEBURST | UDMA_ATTR_ALTSELECT
|
                                    (UDMA_ATTR_HIGH_PRIORITY |
                                    UDMA_ATTR_REQMASK));

    // Configure the control parameters for the SW channel.  The SW channel
    // will be used to transfer between two memory buffers, 32 bits at a
time,
    // and the address increment is 32 bits for both source and destination.
    // The arbitration size will be set to 8, which causes the uDMA
controller
    // to rearbitrate after 8 items are transferred.  This keeps this channel
from
    // hogging the uDMA controller once the transfer is started, and allows
other
    // channels to get serviced if they are higher priority.
    uDMAChannelControlSet(UDMA_CHANNEL_SW | UDMA_PRI_SELECT,
                              UDMA_SIZE_32 | UDMA_SRC_INC_32 |
UDMA_DST_INC_32 |
                              UDMA_ARB_8);

    // Set up the transfer parameters for the software channel.  This will
    // configure the transfer buffers and the transfer size.  Auto mode must
be
    // used for software transfers.
    uDMAChannelTransferSet(UDMA_CHANNEL_SW | UDMA_PRI_SELECT,
                               UDMA_MODE_AUTO, g_ui32SrcBuf, g_ui32DstBuf,
                               MEM_BUFFER_SIZE);

    // Now the software channel is primed to start a transfer.  The channel
    // must be enabled.  For software based transfers, a request must be
    // issued.  After this, the uDMA memory transfer begins.
    uDMAChannelEnable(UDMA_CHANNEL_SW);
    uDMAChannelRequest(UDMA_CHANNEL_SW);
}

int
main(void)
{

    //Configure peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    //Setup for ADC
    ADCHardwareOversampleConfigure(ADC0_BASE, 32);
    ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_TS);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_TS|ADC_CTL_IE|
ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 2);

    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
//enable pin for LED PF2

    IntMasterEnable(); //enable processor interrupts

    FPULazyStackingEnable();

    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                       SYSCTL_XTAL_16MHZ);

    SysCtlPeripheralClockGating(true);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UDMA);

    IntEnable(INT_UDMAERR);
    uDMAEnable();

    uDMAControlBaseSet(pui8ControlTable);

    InitSWTransfer();

    UARTprintf("Enter the cmd: \n"
                    "R: Red LED, \n"
                    "G: Green LED, \n"
                    "B: Blue LED, \n"
                    "T: Temperature, \n"
                    "S: status of the LEDs. \n");

    while(1)
    {
    }
}
```

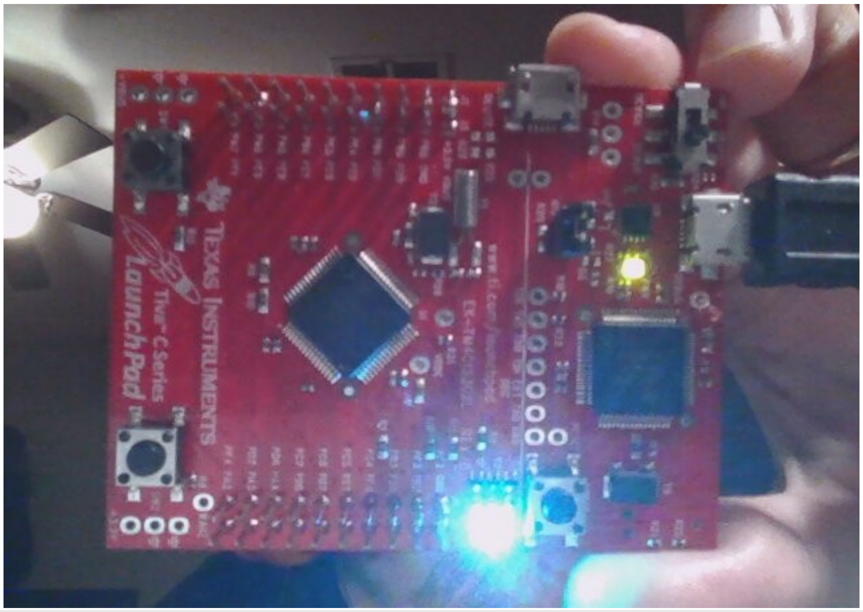
2. Block diagram and/or Schematics showing the components, pins used, and interface.

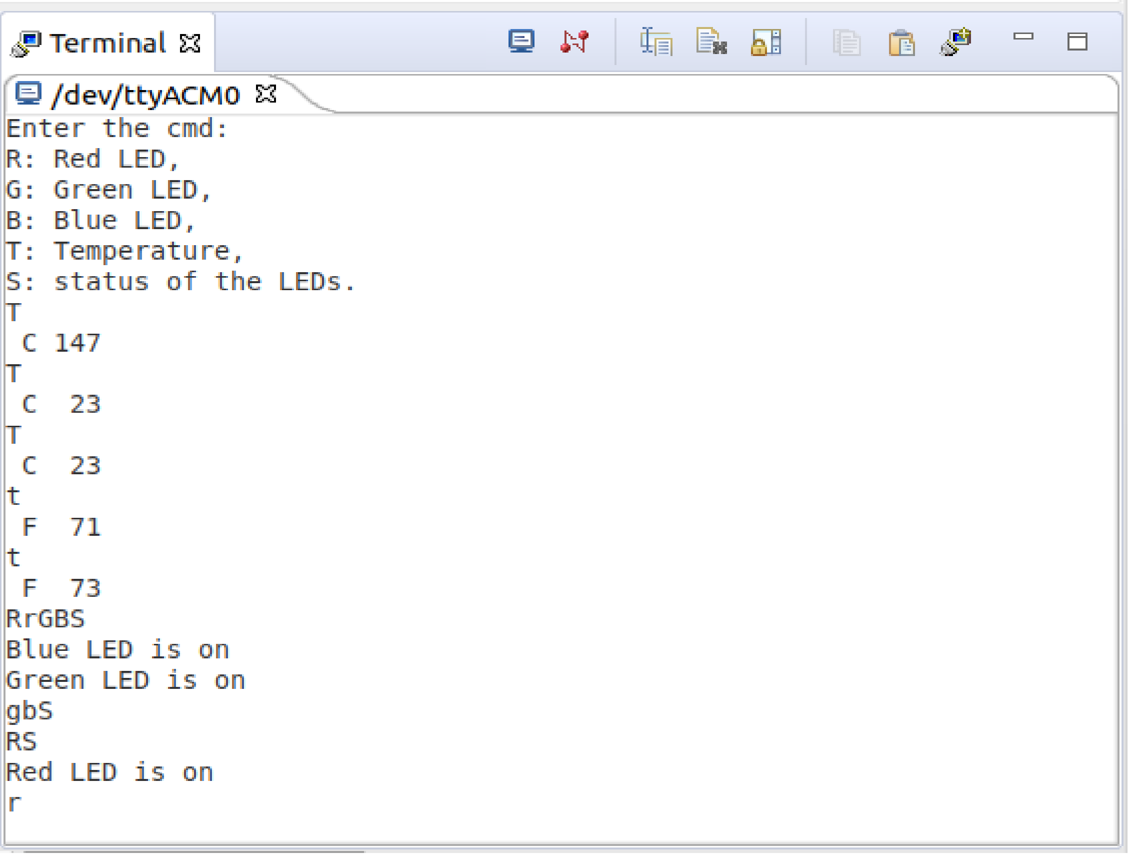


Connected to USB

GPIO_PIN_1,2,3 to show LED from uDMA Command

3. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.





```
Enter the cmd:
R: Red LED,
G: Green LED,
B: Blue LED,
T: Temperature,
S: status of the LEDs.
T
 C 147
T
 C  23
T
 C  23
t
 F  71
t
 F  73
RrGBS
Blue LED is on
Green LED is on
gbS
RS
Red LED is on
r
```

4. Declaration

I understand the Student Academic Misconduct Policy - http://studentconduct.unlv.edu/misconduct/policy.html

“This assignment submission is my own, original work”.
Jenifer Christina